



Tema 4. Plantillas

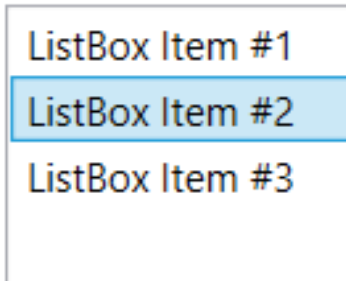
Desarrollo de Interfaces
DAM – IES Doctor Balmis

INDICE

1. El control ListBox
2. Binding en ListBox
3. Plantillas de datos
4. Plantillas de controles
5. Plantilla de control por defecto
6. UserControl
7. Toolkits

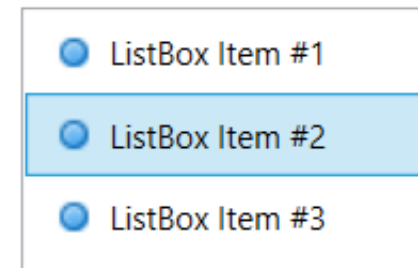
1. El control ListBox

```
<ListBox>
  <ListBoxItem>ListBox Item #1</ListBoxItem>
  <ListBoxItem>ListBox Item #2</ListBoxItem>
  <ListBoxItem>ListBox Item #3</ListBoxItem>
</ListBox>
```



Es Content Control

```
<ListBox>
  <ListBoxItem>
    <StackPanel Orientation="Horizontal">
      <Image Source="bullet.png" />
      <TextBlock VerticalAlignment="Center">ListBox Item #1</TextBlock>
    </StackPanel>
  </ListBoxItem>
  <ListBoxItem>
    <StackPanel Orientation="Horizontal">
      <Image Source="bullet.png" />
      <TextBlock VerticalAlignment="Center">ListBox Item #2</TextBlock>
    </StackPanel>
  </ListBoxItem>
</ListBox>
```



2. Binding en ListBox

Enlace a una lista

```
<ListBox x:Name="EjemploListBox" ItemsSource="{Binding}" />
```

Item #1
Item #2
Item #3

```
public MainWindow()  
{  
    InitializeComponent();  
  
    List<string> lista = new List<string> { "Item #1", "Item #2", "Item #3"};  
    EjemploListBox.DataContext = lista;  
}
```

2. Binding en ListBox

Enlace a una ObservableCollection

```
<ListBox x:Name="EjemploListBox" ItemsSource="{Binding}" />
```

Item #1
Item #2
Item #3

```
public MainWindow()  
{  
    InitializeComponent();  
  
    ObservableCollection<string> lista = new ObservableCollection<string> { "Item #1", "Item #2", "Item #3"};  
    EjemploListBox.DataContext = lista;  
}
```

**Lista que implementa la interfaz
*INotifyCollectionChanged***

2. Binding en ListBox

Enlace a objetos de negocio

```
<ListBox x:Name="EjemploListBox" ItemsSource="{Binding}"
        DisplayMemberPath="Nombre" SelectedValuePath="Edad"/>
```

```
public MainWindow()
{
    InitializeComponent();

    ObservableCollection<Persona> lista = new ObservableCollection<Persona>();
    lista.Add(new Persona("Juan", 32));
    lista.Add(new Persona("Antonio", 28));
    lista.Add(new Persona("Ana", 27));

    EjemploListBox.DataContext = lista;
}
```

3. Plantillas de datos

```
<ListBox x:Name="EjemploListBox" ItemsSource="{Binding}" SelectedValuePath="Edad">
    <ListBox.ItemTemplate>
        <DataTemplate>
            <StackPanel Orientation="Horizontal">
                <TextBlock Text="{Binding Nombre}" />
                <TextBlock Text=" - " />
                <TextBlock Text="{Binding Edad}" />
                <TextBlock Text=" años" />
            </StackPanel>
        </DataTemplate>
    </ListBox.ItemTemplate>
</ListBox>
```

Juan - 32 años
Antonio - 28 años
Ana - 27 años

3. Plantillas de datos

TextBlock con Run

```
<ListBox x:Name="EjemploListBox" ItemsSource="{Binding}" SelectedValuePath="Edad">
    <ListBox.ItemTemplate>
        <DataTemplate>
            <TextBlock>
                <Run Text="{Binding Nombre}" />
                <Run Text=" - " />
                <Run Text="{Binding Edad}" />
                <Run Text=" años" />
            </TextBlock>
        </DataTemplate>
    </ListBox.ItemTemplate>
</ListBox>
```

Juan - 32 años
Antonio - 28 años
Ana - 27 años

3. Plantillas de datos

Plantilla en recursos

```
<Window.Resources>
    <DataTemplate x:Key="plantilla">
        <TextBlock>
            <Run Text="{Binding Nombre}" />
            <Run Text=" - " />
            <Run Text="{Binding Edad}" />
            <Run Text=" años" />
        </TextBlock>
    </DataTemplate>
</Window.Resources>

<ListBox x:Name="EjemploListBox" ItemsSource="{Binding}" SelectedValuePath="Edad"
    ItemTemplate="{StaticResource plantilla}"/>
```

3. Plantillas de datos

Plantilla implícita

```
<Window.Resources>
  <DataTemplate DataType="{x:Type local:Persona}">
    <TextBlock>
      <Run Text="{Binding Nombre}" />
      <Run Text=" - " />
      <Run Text="{Binding Edad}" />
      <Run Text=" años" />
    </TextBlock>
  </DataTemplate>
</Window.Resources>
```

Asociamos la plantilla a una clase de negocio

```
<ListBox Name="EjemploListBox" ItemsSource="{Binding}" SelectedValuePath="Edad" />
```

4. Plantillas de controles

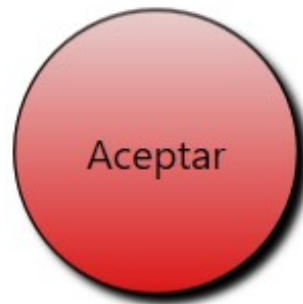


4. Plantillas de controles

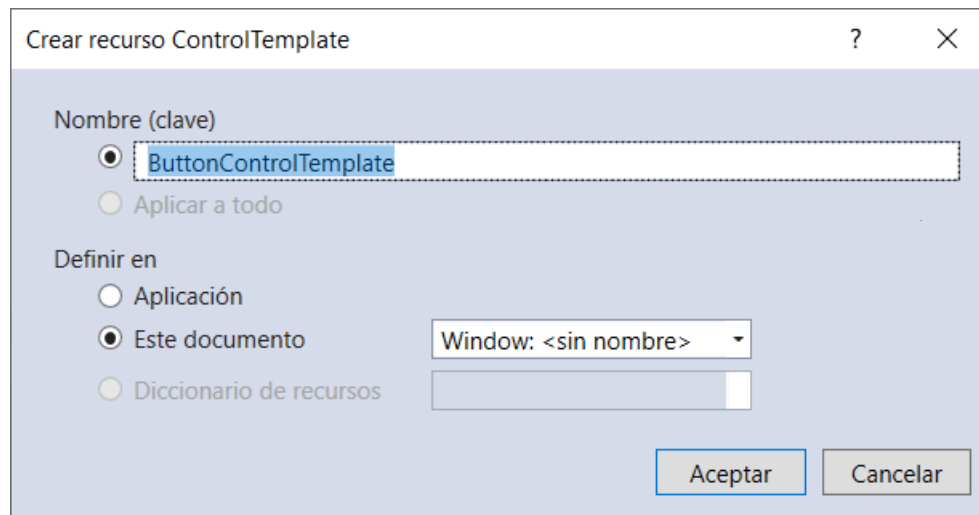
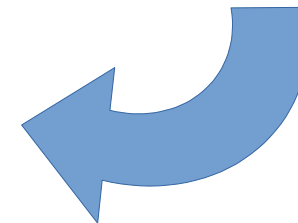
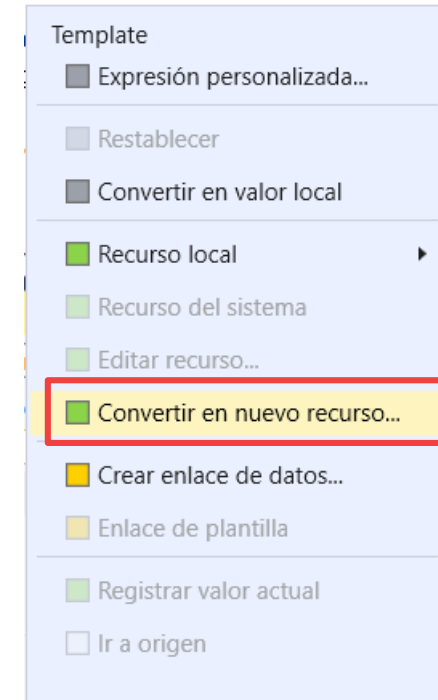
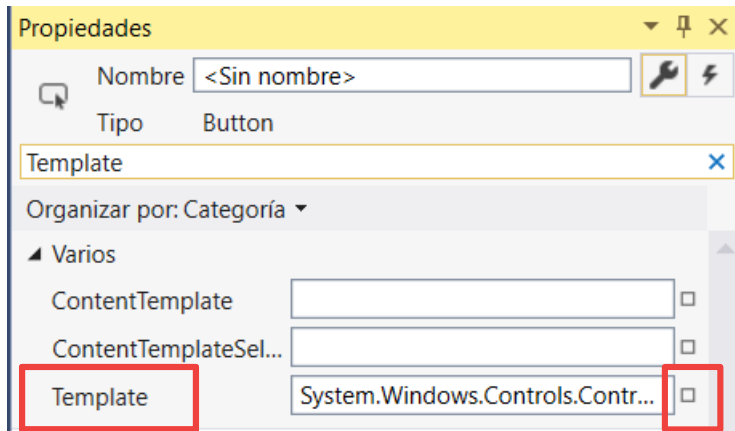
```
<Window.Resources>
  <ControlTemplate TargetType="Button" x:Key="miBoton">
    <Grid>
      <Ellipse Height="{TemplateBinding Height}" Stroke="Black" Width="{TemplateBinding Height}">
        <Ellipse.Effect>
          <DropShadowEffect/>
        </Ellipse.Effect>
        <Ellipse.Fill>
          <LinearGradientBrush EndPoint="0.5,1" StartPoint="0.5,0">
            <GradientStop Color="#FFE8CFCF" Offset="0"/>
            <GradientStop Color="#FFDC1818" Offset="1"/>
          </LinearGradientBrush>
        </Ellipse.Fill>
      </Ellipse>
      <ContentPresenter VerticalAlignment="Center" HorizontalAlignment="Center"></ContentPresenter>
    </Grid>
  </ControlTemplate>
</Window.Resources>
```

4. Plantillas de controles

```
<Button Template="{StaticResource miBoton}" Content="Aceptar" Height="112" Width="225" FontSize="16" />
```



5. Plantilla de control por defecto



6. UserControl

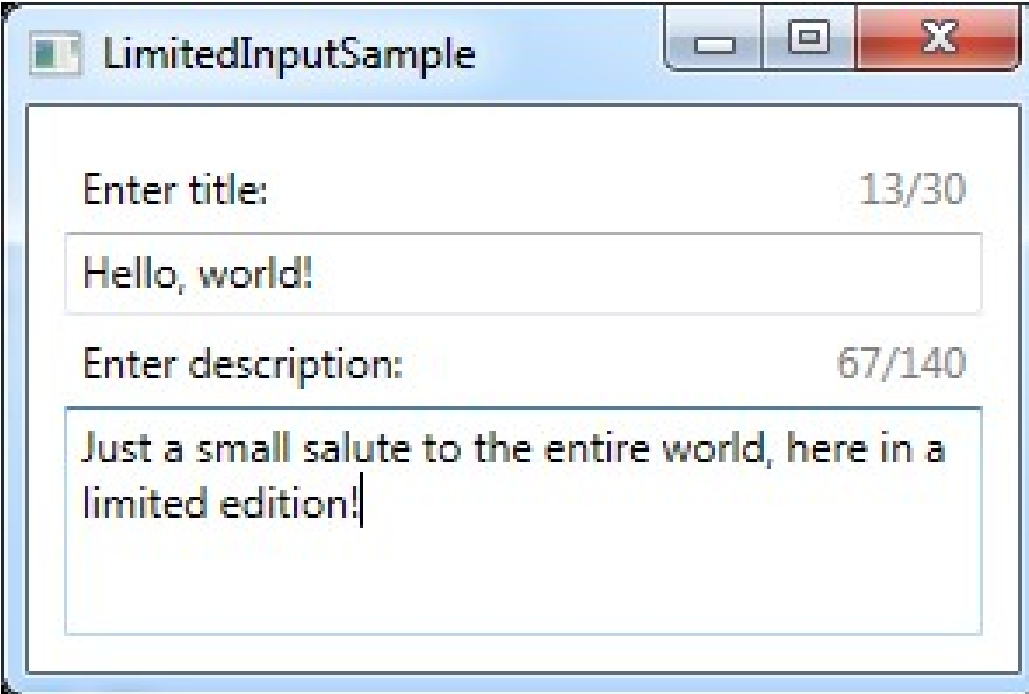
CustomControl

- Extiende un control existente
- Están formados por código y un estilo por defecto
- Se les pueden aplicar estilos y plantillas

UserControl

- Agrupa varios controles existentes
- Están formados por XAML y código
- No se les pueden aplicar estilos o plantillas

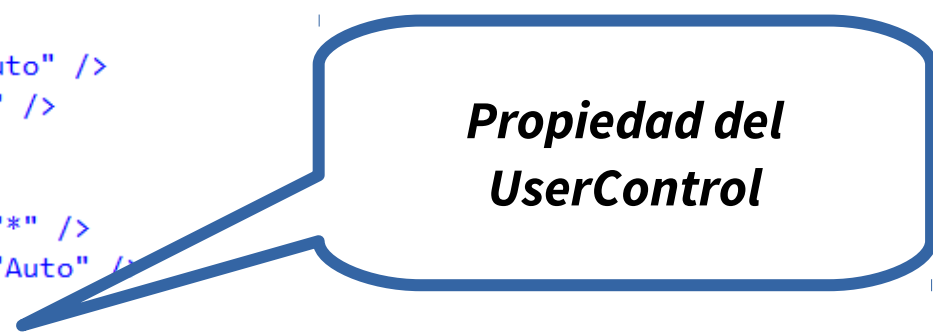
6. UserControl



The screenshot shows a standard Windows application window with the title bar 'LimitedInputSample'. Inside the window, there are two text input fields. The first field is preceded by the label 'Enter title:' and has a character count '13/30' to its right. The text 'Hello, world!' is entered into this field. The second field is preceded by the label 'Enter description:' and has a character count '67/140' to its right. The text 'Just a small salute to the entire world, here in a limited edition!' is entered into this field. The window has standard minimize, maximize, and close buttons in the title bar.

6. UserControl (XAML)

```
<UserControl x:Class="UserControlSample.LimitedInputControl"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
    xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
    xmlns:local="clr-namespace:UserControlSample"
    mc:Ignorable="d"
    d:DesignHeight="450" d:DesignWidth="800">
    <Grid>
        <Grid.RowDefinitions>
            <RowDefinition Height="Auto" />
            <RowDefinition Height="*" />
        </Grid.RowDefinitions>
        <Grid.ColumnDefinitions>
            <ColumnDefinition Width="*" />
            <ColumnDefinition Width="Auto" />
        </Grid.ColumnDefinitions>
        <Label Content="{Binding Title}" />
        <Label Grid.Column="1">
            <StackPanel Orientation="Horizontal">
                <TextBlock Text="{Binding ElementName=LimitedTetBox, Path=Text.Length}" />
                <TextBlock Text="/" />
                <TextBlock Text="{Binding MaxLength}" />
            </StackPanel>
        </Label>
        <TextBox x:Name="LimitedTetBox" Text="{Binding Text, UpdateSourceTrigger=PropertyChanged}"
            MaxLength="{Binding MaxLength}" Grid.Row="1" Grid.ColumnSpan="2"/>
    </Grid>
</UserControl>
```




**Propiedad del
UserControl**

6. UserControl (code behind)

```
namespace UserControlSample
{
    public partial class LimitedInputControl : UserControl
    {
        public LimitedInputControl()
        {
            InitializeComponent();
            this.DataContext = this;
        }

        public string Title
        {
            get { return (string)GetValue(TitleProperty); }
            set { SetValue(TitleProperty, value); }
        }

        public static readonly DependencyProperty TitleProperty =
            DependencyProperty.Register("Title", typeof(string), typeof(LimitedInputControl), new PropertyMetadata(""));
    }
}
```



***Propiedad de dependencia
(propdp en Visual Studio)***

6. UserControl (code behind)

```
public int MaxLength
{
    get { return (int)GetValue(MaxLengthProperty); }
    set { SetValue(MaxLengthProperty, value); }
}
```

Tipo del UserControl

```
public static readonly DependencyProperty MaxLengthProperty =
DependencyProperty.Register("MaxLength", typeof(int), typeof(LimitedInputControl), new PropertyMetadata(0));
```

Nombre de la propiedad

Tipo de la propiedad

Valor por defecto

```
public string Text
{
    get { return (string)GetValue(TextProperty); }
    set { SetValue(TextProperty, value); }
}
```

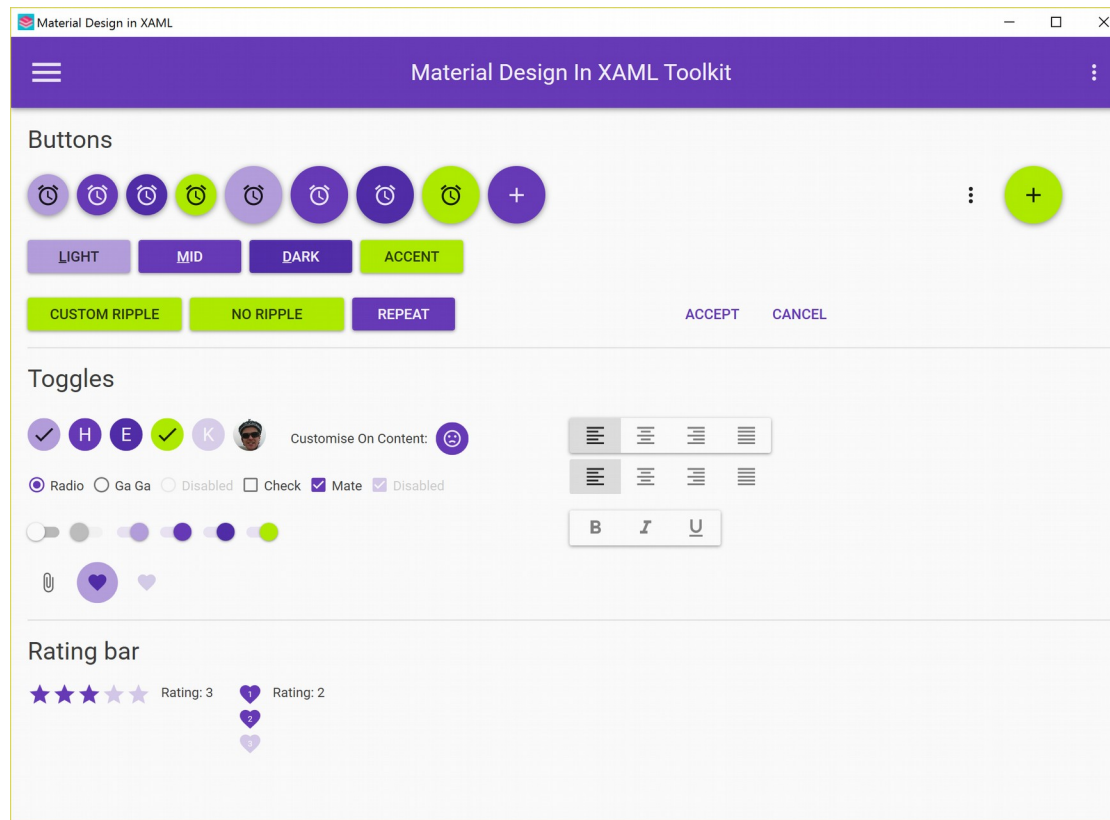
```
public static readonly DependencyProperty TextProperty =
DependencyProperty.Register("Text", typeof(string), typeof(LimitedInputControl), new PropertyMetadata(""));
```

6. UserControl (uso)

```
<Window x:Class="UserControlSample.MainWindow"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
        xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
        xmlns:local="clr-namespace:UserControlSample"
        mc:Ignorable="d"
        Title="MainWindow" Height="450" Width="800">
    <StackPanel>
        <local:LimitedInputControl x:Name="UsuarioLimitedInputControl"
                                MaxLength="200" Title="Usuario">
        </local:LimitedInputControl>
        <TextBlock Text="{Binding ElementName=UsuarioLimitedInputControl,Path=Text}" />
    </StackPanel>
</Window>
```

7. Toolkits

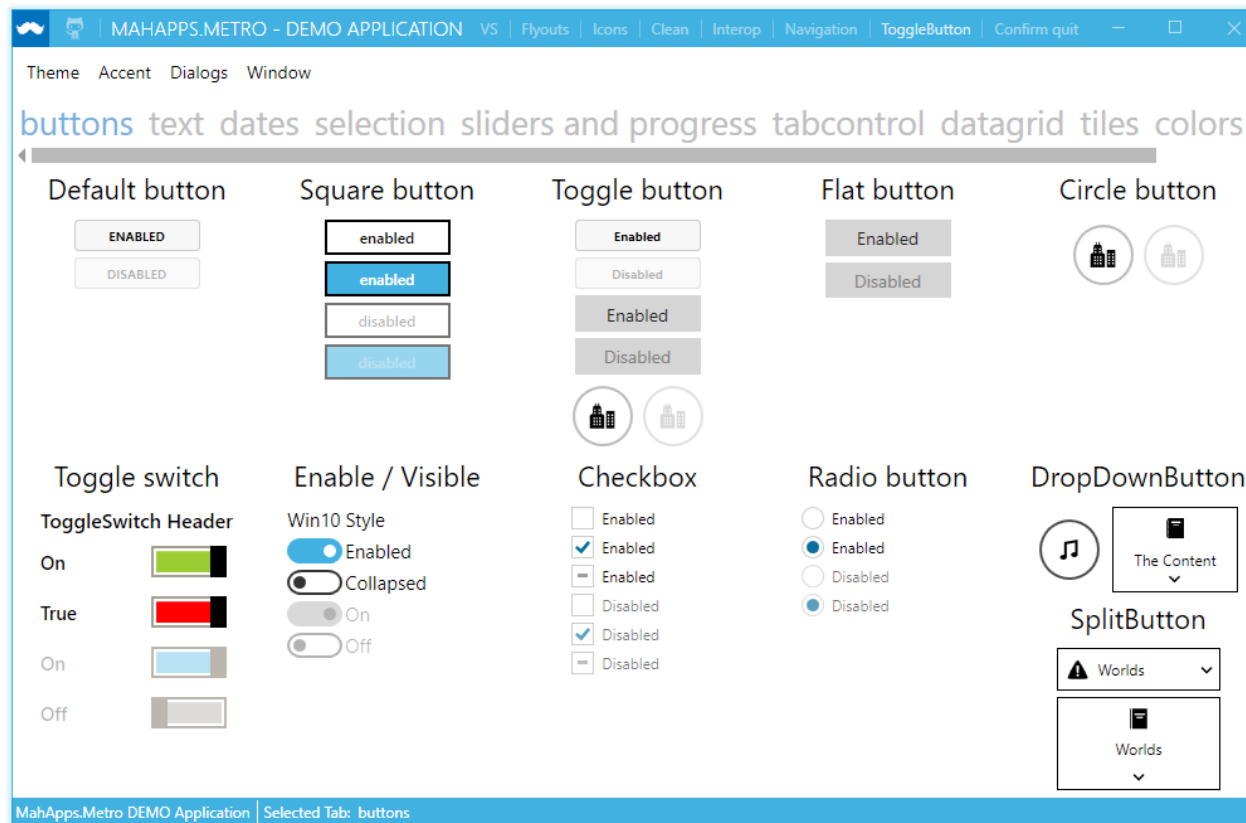
Material Design in XAML



<http://materialdesigninxaml.net/>

7. Toolkits

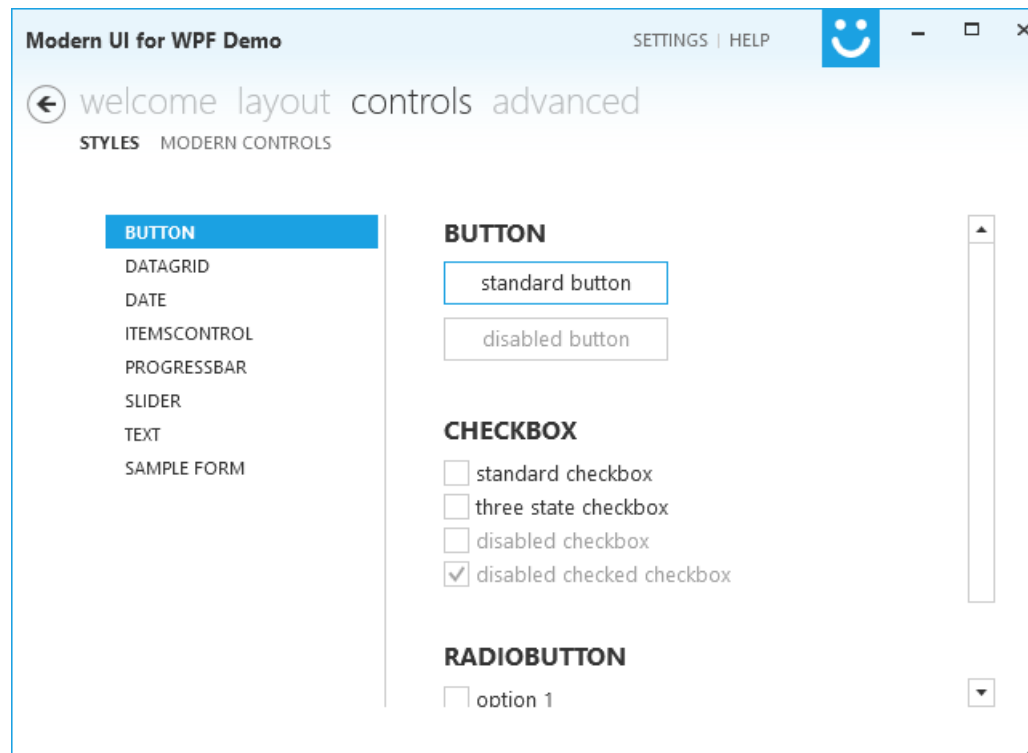
mahapps.metro



<https://mahapps.com/>

7. Toolkits

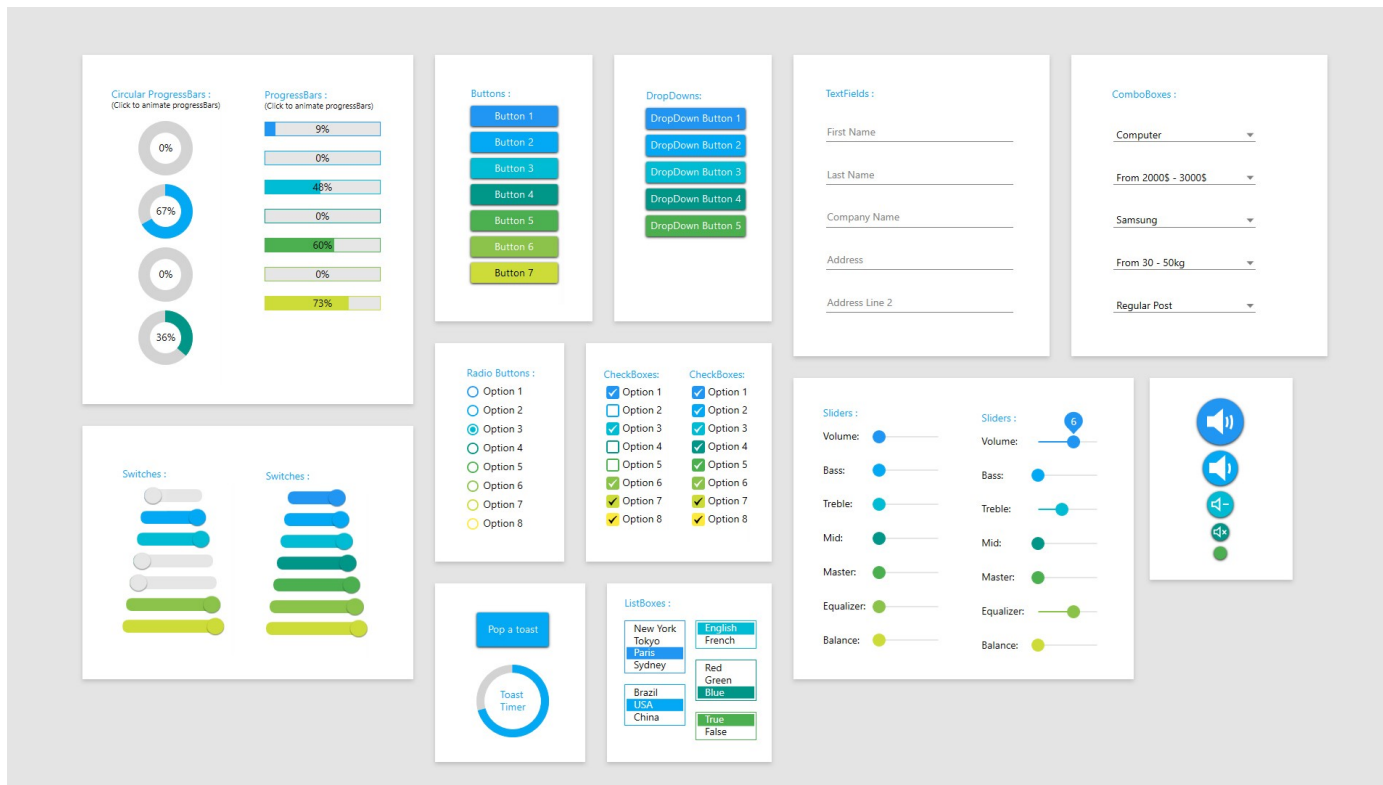
Modern UI for WPF



<https://github.com/firstfloorsoftware/mui>

7. Toolkits

Extended WPF Toolkit



<https://github.com/xceedsoftware/wpftoolkit>



Tema 4. Plantillas

Desarrollo de Interfaces
DAM – IES Doctor Balmis



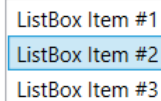
Javier Catalá

INDICE

1. El control ListBox
2. Binding en ListBox
3. Plantillas de datos
4. Plantillas de controles
5. Plantilla de control por defecto
6. UserControl
7. Toolkits

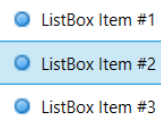
1. El control ListBox

```
<ListBox>
  <ListBoxItem>ListBox Item #1</ListBoxItem>
  <ListBoxItem>ListBox Item #2</ListBoxItem>
  <ListBoxItem>ListBox Item #3</ListBoxItem>
</ListBox>
```



Es Content Control

```
<ListBox>
  <ListBoxItem>
    <StackPanel Orientation="Horizontal">
      <Image Source="bullet.png" />
      <TextBlock VerticalAlignment="Center">ListBox Item #1</TextBlock>
    </StackPanel>
  </ListBoxItem>
  <ListBoxItem>
    <StackPanel Orientation="Horizontal">
      <Image Source="bullet.png" />
      <TextBlock VerticalAlignment="Center">ListBox Item #2</TextBlock>
    </StackPanel>
  </ListBoxItem>
</ListBox>
```



El control *ListBox* muestra al usuario una lista de items, permitiéndole seleccionar uno o varios de ellos (dependiendo del valor de la propiedad *SelectionMode*).

Una de las opciones que tenemos para especificar los items del listado es utilizar el elemento anidado *ListBoxItem*. Este elemento es un content control, por lo que puede consistir en un simple texto (como en el primer ejemplo de la diapositiva) o utilizar un contenedor para darle una estructura visual más compleja a cada item (como en el segundo ejemplo).

Existen otros controles similares al *ListBox*, como el *ItemsControl* (un listado sin funcionalidad de selección) o el *ComboBox* (que incorpora la funcionalidad de despliegue del listado).

2. Binding en ListBox

Enlace a una lista

```
<ListBox x:Name="EjemploListBox" ItemsSource="{Binding}" />
```



Item #1
Item #2
Item #3

```
public MainWindow()  
{  
    InitializeComponent();  
  
    List<string> lista = new List<string> { "Item #1", "Item #2", "Item #3"};  
    EjemploListBox.DataContext = lista;  
}
```

Otra opción que ofrece WPF para indicar los elementos que se tendrán que mostrar en el listado es la de enlazar el control a una lista.

Como se puede ver en el ejemplo, en el constructor de la ventana creamos una nueva lista de cadenas, y establecemos esta lista como *DataContext* del *ListBox*.

El *ListBox* cuenta con la propiedad *ItemsSource* para indicar el origen de los items, que es la que deberemos enlazar con la lista. Como el valor que queremos dar a la propiedad *ItemsSource* es la propia lista, no tendremos que indicar ninguna propiedad en el *binding*.

2. Binding en ListBox

Enlace a una ObservableCollection

```
<ListBox x:Name="EjemploListBox" ItemsSource="{Binding}" />
```

```
public MainWindow()  
{  
    InitializeComponent();  
  
    ObservableCollection<string> lista = new ObservableCollection<string> { "Item #1", "Item #2", "Item #3"};  
    EjemploListBox.DataContext = lista;  
}
```

Item #1
Item #2
Item #3

Lista que implementa la interfaz *INotifyCollectionChanged*

Tema 4. Plantillas

5

Cuando enlazamos un ListBox a una lista, podemos encontrarnos con un problema. Si actualizamos la lista, es muy probable que los cambios no se vean reflejados en la interfaz.

De forma análoga a lo que ocurre con el cambio en las propiedades de un objeto (necesitamos la interfaz *INotifyPropertyChanged* para informar de los cambios), para informar de los cambios en el contenido de una lista tenemos que implementar la interfaz *INotifyCollectionChanged*.

Tenemos a nuestra disposición en el espacio de nombres *System.Collections.ObjectModel* el tipo de colección *ObservableCollection*, que ya implementa dicha interfaz.

2. Binding en ListBox

Enlace a objetos de negocio

```
<ListBox x:Name="EjemploListBox" ItemsSource="{Binding}"
        DisplayMemberPath="Nombre" SelectedValuePath="Edad"/>
```

```
public MainWindow()
{
    InitializeComponent();

    ObservableCollection<Persona> lista = new ObservableCollection<Persona>();
    lista.Add(new Persona("Juan", 32));
    lista.Add(new Persona("Antonio", 28));
    lista.Add(new Persona("Ana", 27));

    EjemploListBox.DataContext = lista;
}
```

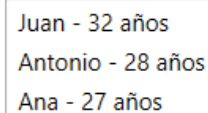
En muchas ocasiones, el *ListBox* deberá estar enlazado a una lista de objetos de negocio. En el ejemplo, se realiza el enlace a una lista de objetos *Persona*.

En este caso, es necesario decidir qué propiedad del objeto se debe mostrar en el *ListBox* (*DisplayMemberPath*) y a qué propiedad del objeto queremos acceder cuando usemos la propiedad *SelectedValue* del *ListBox*.

El *ListBox* también ofrece la propiedad *SelectedItem*, que nos permitirá acceder al objeto de negocio completo que se haya seleccionado.

3. Plantillas de datos

```
<ListBox x:Name="EjemploListBox" ItemsSource="{Binding}" SelectedValuePath="Edad">
  <ListBox.ItemTemplate>
    <DataTemplate>
      <StackPanel Orientation="Horizontal">
        <TextBlock Text="{Binding Nombre}" />
        <TextBlock Text=" - " />
        <TextBlock Text="{Binding Edad}" />
        <TextBlock Text=" años" />
      </StackPanel>
    </DataTemplate>
  </ListBox.ItemTemplate>
</ListBox>
```



Juan - 32 años
Antonio - 28 años
Ana - 27 años

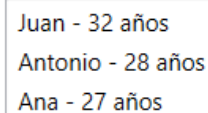
Las plantillas de datos son una herramienta que nos ofrece WPF para definir la estructura visual que deben tener los distintos items de un control de lista, como un *ListBox*. Son especialmente útiles cuando estamos enlazando el control a un listado de objetos, y queremos utilizar varias propiedades del objeto en la composición visual del item.

En el ejemplo, se define una plantilla para cada item (propiedad *ItemTemplate*) que consiste en una plantilla de datos (*DataTemplate*) compuesta por un *StackPanel* horizontal con cuatro *TextBlock*. De esta forma, en cada item de la lista podremos mostrar dos propiedades del objeto enlazado, en este caso el nombre y la edad.

3. Plantillas de datos

TextBlock con Run

```
<ListBox x:Name="EjemploListBox" ItemsSource="{Binding}" SelectedValuePath="Edad">
  <ListBox.ItemTemplate>
    <DataTemplate>
      <TextBlock>
        <Run Text="{Binding Nombre}" />
        <Run Text=" - " />
        <Run Text="{Binding Edad}" />
        <Run Text=" años" />
      </TextBlock>
    </DataTemplate>
  </ListBox.ItemTemplate>
</ListBox>
```



Juan - 32 años
Antonio - 28 años
Ana - 27 años

En esta diapositiva se muestra una alternativa al ejemplo anterior, usando una funcionalidad que ofrece el control *TextBlock*. El texto que se muestra en este control se puede subdividir en partes utilizando el elemento *Run*.

Cada una de las partes puede enlazarse de forma independiente. Además, cada elemento *Run* puede configurarse en cuestiones como tipo de letra, color, tamaño de la fuente,...

Esta posibilidad del *TextBlock* no está disponible cuando usamos el control *Label*.

3. Plantillas de datos

Plantilla en recursos

```
<Window.Resources>
  <DataTemplate x:Key="plantilla">
    <TextBlock>
      <Run Text="{Binding Nombre}" />
      <Run Text=" - " />
      <Run Text="{Binding Edad}" />
      <Run Text=" años" />
    </TextBlock>
  </DataTemplate>
</Window.Resources>

<ListBox x:Name="EjemploListBox" ItemsSource="{Binding}" SelectedValuePath="Edad"
  ItemTemplate="{StaticResource plantilla}"/>
```

Cuando las aplicaciones van creciendo, es muy probable que tengamos que reutilizar la misma plantilla de datos en diferentes controles.

Para organizar mejor el código y evitar duplicidades, podemos alojar las plantillas en los recursos (de la ventana, de la aplicación o en un diccionario de recursos, como vimos con los estilos).

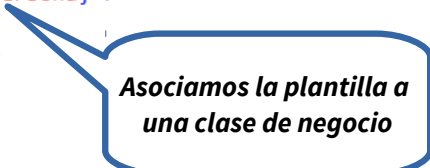
Cuando queramos utilizar dicha plantilla, simplemente haremos referencia a ella en la propiedad *ItemTemplate* del *ListBox*, como se puede ver en el ejemplo.

3. Plantillas de datos

Plantilla implícita

```
<Window.Resources>
  <DataTemplate DataType="{x:Type local:Persona}">
    <TextBlock>
      <Run Text="{Binding Nombre}" />
      <Run Text=" - " />
      <Run Text="{Binding Edad}" />
      <Run Text=" años" />
    </TextBlock>
  </DataTemplate>
</Window.Resources>

<ListBox Name="EjemploListBox" ItemsSource="{Binding}" SelectedValuePath="Edad" />
```

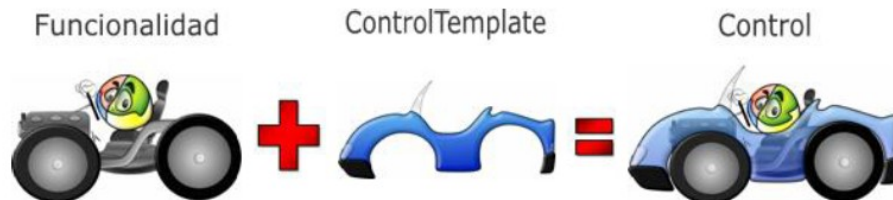


Asociamos la plantilla a una clase de negocio

Una última posibilidad que nos ofrecen las plantillas de datos es la de asociarlas en su definición a una clase de negocio, mediante la propiedad *DataType* de la plantilla. Es lo que se denomina plantilla implícita.

De esta forma, cuando se realice un enlace a un objeto de dicha clase, se utilizará de forma implícita la plantilla de datos, sin necesidad de indicarlo con la propiedad *ItemTemplate* del *ListBox*.

4. Plantillas de controles



En este apartado vamos a hablar de las plantillas de los controles. La plantilla de un control es la responsable de definir el aspecto visual del control. WPF nos permite modificar por completo la plantilla de un control manteniendo su funcionalidad original.

Es importante destacar que, a pesar de la similitud en el nombre, no existe ninguna relación entre las plantillas de datos y de controles. Mientras las primeras solo aplican a controles de tipo lista (como el *ListBox*) y solo se utilizan para definir el aspecto visual de los items, las plantillas de control están presentes en todos los controles y definen el aspecto visual del propio control.

4. Plantillas de controles

```
<Window.Resources>
  <ControlTemplate TargetType="Button" x:Key="miBoton">
    <Grid>
      <Ellipse Height="{TemplateBinding Height}" Stroke="Black" Width="{TemplateBinding Height}">
        <Ellipse.Effect>
          <DropShadowEffect/>
        </Ellipse.Effect>
        <Ellipse.Fill>
          <LinearGradientBrush EndPoint="0.5,1" StartPoint="0.5,0">
            <GradientStop Color="#FFE8CFCF" Offset="0"/>
            <GradientStop Color="#FFDC1818" Offset="1"/>
          </LinearGradientBrush>
        </Ellipse.Fill>
      </Ellipse>
      <ContentPresenter VerticalAlignment="Center" HorizontalAlignment="Center"></ContentPresenter>
    </Grid>
  </ControlTemplate>
</Window.Resources>
```

En el ejemplo se muestra una plantilla de control (*ControlTemplate*) definida en los recursos de la ventana. Como se puede apreciar, se asocia a un tipo de control (*TargetType*) y se le asigna un identificador (*x:Key*).

El ejemplo define una nueva plantilla para un botón. Se utiliza una elipse dentro de un *Grid*, y superpuesto sobre la elipse un *ContentPresenter*. Este control representa al valor que se da a la propiedad *Content* del botón.

También se puede observar que algunas propiedades están utilizando un tipo de enlace especial llamado *TemplateBinding*. Estos enlaces hacen referencia a propiedades del control que está utilizando esta plantilla.

4. Plantillas de controles

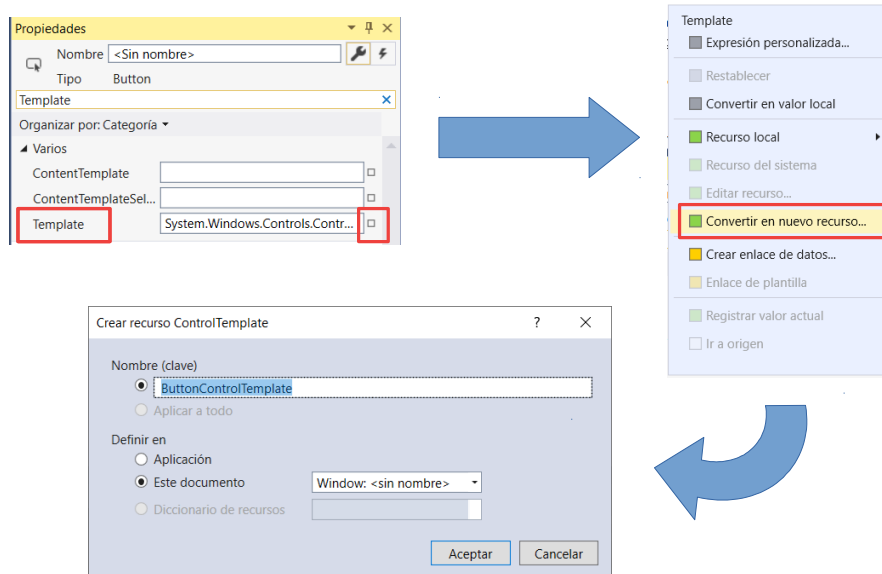
```
<Button Template="{StaticResource miBoton}" Content="Aceptar" Height="112" Width="225" FontSize="16" />
```



Para aplicar la plantilla definida a un control utilizamos la propiedad *Template*.

Como vemos en el ejemplo, la apariencia visual del botón ha cambiado por completo, pero su funcionalidad seguirá siendo exactamente la misma.

5. Plantilla de control por defecto



En muchas ocasiones, en lugar de definir por completo una nueva plantilla para un control, es más práctico realizar modificaciones sobre la plantilla por defecto de ese control.

Para ello, es necesario seguir el procedimiento indicado en la diapositiva:

1. Vamos a la propiedad *Template* en el panel de Propiedades, y le damos al selector con forma de cuadrado.
2. Pulsamos en convertir en nuevo recurso.
3. Le damos un nombre a la plantilla (*x:Key*) y decidimos dónde queremos situarla (*App.xaml*, *Window.Resources* o en el propio control).

Tras esto, ya podremos modificar la plantilla creada.

6. UserControl

CustomControl

- Extiende un control existente
- Están formados por código y un estilo por defecto
- Se les pueden aplicar estilos y plantillas

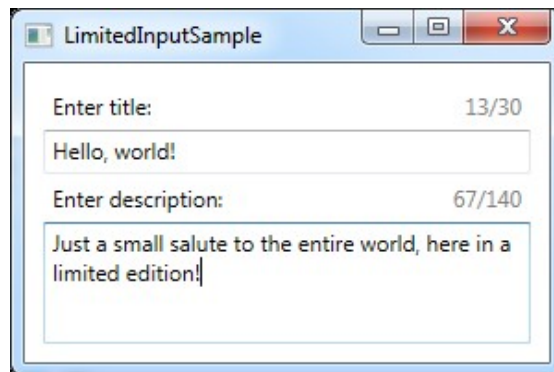
UserControl

- Agrupa varios controles existentes
- Están formados por XAML y código
- No se les pueden aplicar estilos o plantillas

En ocasiones podemos necesitar una funcionalidad que no está cubierta por ninguno de los controles que ofrece por defecto WPF. Podemos crear nuestros propios controles, y para ello disponemos de dos mecanismos:

- *CustomControl*: crear un control nuevo a partir de uno existente. A todos los efectos se comportan como el resto de controles.
- *UserControl*: agrupar un conjunto de controles y utilizarlos de forma conjunta como un nuevo control. Están más limitados que los *CustomControl*, pero son más sencillos de utilizar.

6. UserControl




Vamos a ver un ejemplo sencillo de un UserControl, consistente en un TextBox con límite de caracteres.

Como se puede ver en la imagen, el control estará compuesto de una etiqueta, el propio TextBox y un indicador del número de caracteres usados y del máximo.

6. UserControl (XAML)

```
<UserControl x:Class="UserControlSample.LimitedInputControl"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
    xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
    xmlns:local="clr-namespace:UserControlSample"
    mc:Ignorable="d"
    d:DesignHeight="450" d:DesignWidth="800">

    <Grid>
        <Grid.RowDefinitions>
            <RowDefinition Height="Auto" />
            <RowDefinition Height="*" />
        </Grid.RowDefinitions>
        <Grid.ColumnDefinitions>
            <ColumnDefinition Width="*" />
            <ColumnDefinition Width="Auto" />
        </Grid.ColumnDefinitions>
        <Label Content="{Binding Title}" />
        <Label Grid.Column="1">
            <StackPanel Orientation="Horizontal">
                <TextBlock Text="{Binding ElementName=LimitedTextBox, Path=Text.Length}" />
                <TextBlock Text="/" />
                <TextBlock Text="{Binding MaxLength}" />
            </StackPanel>
        </Label>
        <TextBox x:Name="LimitedTextBox" Text="{Binding Text, UpdateSourceTrigger=PropertyChanged}"
            MaxLength="{Binding MaxLength}" Grid.Row="1" Grid.ColumnSpan="2"/>
    </Grid>
</UserControl>
```



**Propiedad del
UserControl**

En primer lugar nos vamos a detener en el XAML del *UserControl*. Como se puede observar, el *UserControl* tiene una estructura muy similar a la de una ventana.

En el XAML se define el layout del control, en este caso un *Grid* de dos filas y dos columnas para situar los distintos elementos del control.


Hay que destacar el uso de algunos binding en el código. Las propiedades a las que se está enlazando (*Title*, *MaxLength* y *Text*), como veremos a continuación, están definidas en el código trasero del *UserControl*.

6. UserControl (code behind)

```
namespace UserControlSample
{
    public partial class LimitedInputControl : UserControl
    {
        public LimitedInputControl()
        {
            InitializeComponent();
            this.DataContext = this;
        }

        public string Title
        {
            get { return (string)GetValue(TitleProperty); }
            set { SetValue(TitleProperty, value); }
        }

        public static readonly DependencyProperty TitleProperty =
            DependencyProperty.Register("Title", typeof(string), typeof(LimitedInputControl), new PropertyMetadata(""));
    }
}
```



Propiedad de dependencia
(propdp en Visual Studio)

En el código trasero del control definimos las propiedades del UserControl:

- *Title*: la etiqueta que describe el campo
- *MaxLength*: tamaño máximo del texto
- *Text*: texto escrito por el usuario

Para poder utilizar estas propiedades como las del resto de controles, y aplicarles los mecanismos de WPF (binding, estilos, triggers,...) debemos crearlas como propiedades de dependencia. Visual Studio dispone del snippet *propdp* para crearlas fácilmente.

Además, para poder hacer referencia a estas propiedades desde el XAML del control, en el constructor se define como *DataContext* del *UserControl* al propio control.

6. UserControl (code behind)

```
public int MaxLength
{
    get { return (int)GetValue(MaxLengthProperty); }
    set { SetValue(MaxLengthProperty, value); }
}

public static readonly DependencyProperty MaxLengthProperty =
DependencyProperty.Register("MaxLength", typeof(int), typeof(LimitedInputControl), new PropertyMetadata(0));
```

Tipo del UserControl

Nombre de la propiedad

Tipo de la propiedad

Valor por defecto

```
public string Text
{
    get { return (string)GetValue(TextProperty); }
    set { SetValue(TextProperty, value); }
}

public static readonly DependencyProperty TextProperty =
DependencyProperty.Register("Text", typeof(string), typeof(LimitedInputControl), new PropertyMetadata(""));
```

En esta diapositiva se muestran el resto de propiedades de dependencia creadas. Como se puede apreciar, al crear la propiedad es necesario indicar:

- Nombre para la propiedad
- Tipo de datos de la propiedad
- Tipo del control que contiene la propiedad
- Valor por defecto para la propiedad

Además de la propiedad de dependencia, se crea una propiedad normal para acceder a ella sin necesidad de utilizar los métodos *GetValue* y *SetValue*.

6. UserControl (uso)

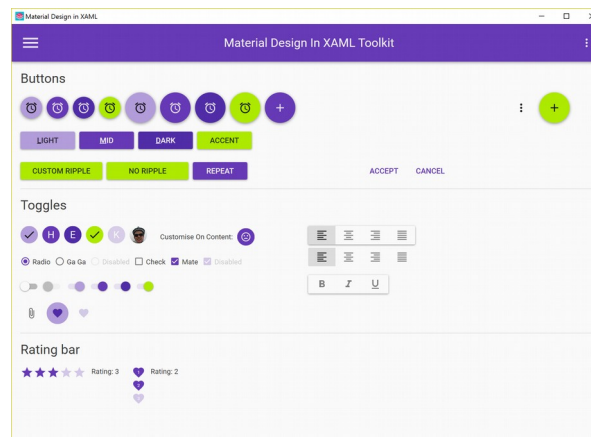
```
<Window x:Class="UserControlSample.MainWindow"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
        xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
        xmlns:local="clr-namespace:UserControlSample"
        mc:Ignorable="d"
        Title="MainWindow" Height="450" Width="800">
    <StackPanel>
        <local:LimitedInputControl x:Name="UsuarioLimitedInputControl"
                                MaxLength="200" Title="Usuario">
        </local:LimitedInputControl>
        <TextBlock Text="{Binding ElementName=UsuarioLimitedInputControl,Path=Text}" />
    </StackPanel>
</Window>
```

Podemos utilizar el control como utilizamos cualquier otro control de WPF. Y podremos darle valor a las propiedades que hayamos definido (además de las propiedades básicas de cualquier control).

Además, como podemos apreciar en el ejemplo, podemos utilizar las propiedades de dependencia creadas en una expresión de binding. También podríamos usarlas en un *setter* de un estilo, o en una condición de un *trigger*.

7. Toolkits

Material Design in XAML



<http://materialdesigninxaml.net/>

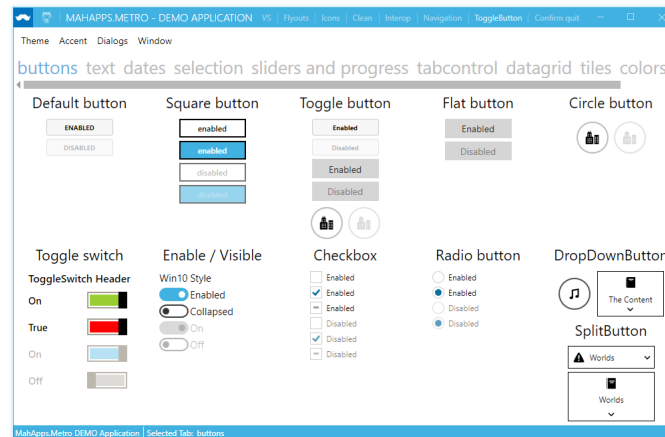
Existen en el mercado conjuntos de controles o toolkits, que se componen de nuevos controles o controles existentes modificados.

Para WPF existen diferentes opciones, algunas comerciales y otras de libre distribución. En las siguientes diapositivas se incluyen algunos ejemplos de toolkits libres para WPF.

Aunque en este tema no vamos a utilizar ningún toolkit, sí lo haremos en el proyecto final que desarrollaremos en la segunda evaluación.

7. Toolkits

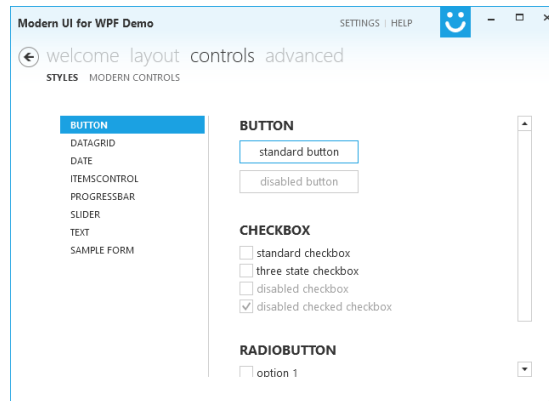
mahapps.metro



<https://mahapps.com/>

7. Toolkits

Modern UI for WPF



<https://github.com/firstfloorsoftware/mui>

7. Toolkits

[illegible]

<https://github.com/xceedsoftware/wpftoolkit>